CSCI 361 Lecture 17: Undecidability Wrap Up & Intro to Complexity Theory

Shikha Singh

Announcements & Logistics

- Pick up **reading assignment # 12**
- **HW 5** graded feedback returned
- **HW 6** deadline extended to tonight at 10 pm
  - Examples in textbook and slides: how to describe TM's
  - Provide the **input**, step-by-step algorithm, and final output (accept/reject)
  - Argue why your reductions are correct: **if and only if** statement
- **HW 7** will be posted today and due next Wed
  - Second to last homework

Looking Ahead

- 5 more lectures left before Thanksgiving break
- 2 more assignments (HW 7 and 8)
- Survey paper logistics
    - Investigate an advanced topic of your interest related to ToC
    - Encouraged to work in pairs but don't have to
    - Will post examples of topics and resources next week
- Google form choosing partner and tentative topic: **Nov 20**
- 1 page draft of background due **Nov 26**
- Class presentation: Dec 5 and short paper (3 pages) due **Dec 6**

Last Time

- Described the computation-history-method to prove undecidability
- Proved that ALL_{CFG} and PCP are undecidable by encoding computation histories

Today

- Wrap up **computability theory**
- Start **complexity theory**
- In the coming weeks:
  - Classes P, NP, EXP
  - P vs NP
  - NP hardness and NP Completeness

Post Correspondence Problem

- An instance of the Post correspondence problem (PCP) is two sequences $A = (a_1, a_2, \ldots, a_m)$ and $B = (b_1, b_2, \ldots, b_m)$ of strings where $a_i, b_i \in \Sigma^*$
- **Problem.** Does there exist a finite sequence $i_1, i_2, \ldots, i_k$ where each i_j is an index from $1, \ldots, m$ such that $a_{i_1} a_{i_2} \ldots a_{i_k} = b_{i_1} b_{i_2} \ldots b_{i_k}$
- **Alternate Formulation:** An input is a collection of dominos each containing two strings $\left[\dfrac{a_1}{b_1}\right], \left[\dfrac{a_2}{b_b}\right], \ldots, \left[\dfrac{a_m}{b_m}\right]$ and the goal is to find a sequence of these dominoes (*repetitions are allowed*) such that the string formed by concatenating the top is the same as the string formed by concatenating the bottom

Post Correspondence Problem

- PCP example: E.g. Consider

$$\left\{ \left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right] \right\}.$$

- A possible solution

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{b}}{\mathtt{ca}}\right]\left[\frac{\mathtt{ca}}{\mathtt{a}}\right]\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{abc}}{\mathtt{c}}\right]$$

Reductions from PCP

- **Theorem. (Last Class)** PCP is undecidable.
- **HW 7 problem:** Reduce PCP to show that

$\cap_{\mathrm{CFG}} = \{\langle G_1, G_2 \rangle \mid G_1, G_2$ are CFGs such that $L(G_1) \cap L(G_2) = \varnothing\}$
is undecidable.

- ***Hint:*** Given PCP instance (A, B), create CFLs L_A and L_B as follows:

$$A \to a_1 A i_1 \mid a_2 A i_2 \mid \cdots \mid a_m A i_m$$
$$A \to a_1 i_1 \mid a_2 i_2 \mid \cdots \mid a_m i_m$$

$$B \to b_1 B i_1 \mid b_2 B i_2 \mid \cdots \mid b_m B i_m$$
$$B \to a_1 i_1 \mid a_2 i_2 \mid \cdots \mid a_m i_m$$

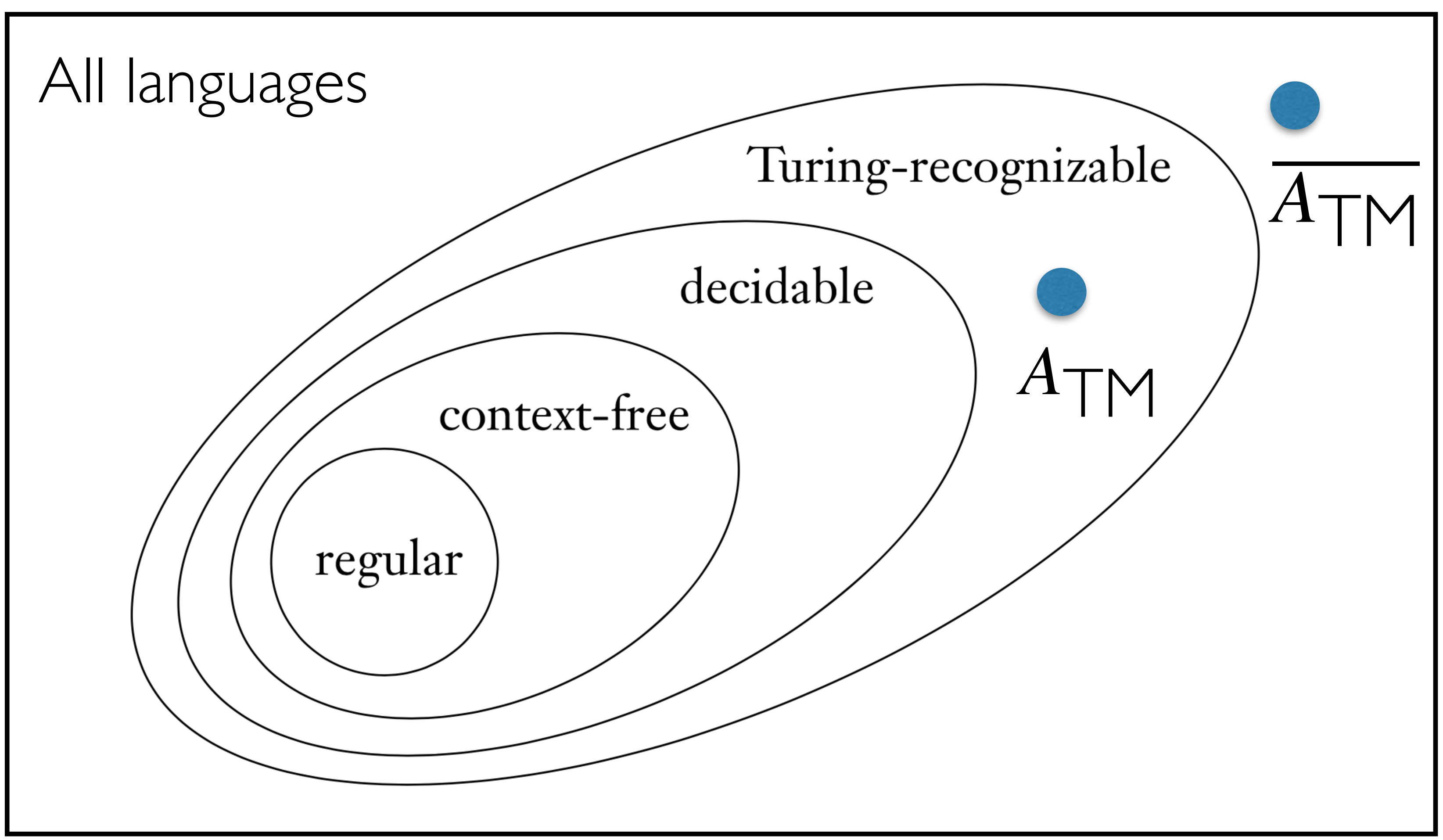
- What strings do they generate? Can we solve PCP using $\cap_{\mathrm{CFG}}$ decider?

Undecidability Takeaways

- Almost all properties of regular languages are decidable
- Lots of undecidable problems about CFGs
  - Let G_1, G_2 be CFGs and R be a regular expression, then the following questions are undecidable:
    - Is $L(G_1) = L(G_2)$?
    - Is $L(G_1) = L(R)$?
    - Is $L(G_1) \subseteq L(G_2)$?
    - Is $L(R) \subseteq L(G_1)$?
- Deciding any non-trivial property of TM is undecidable
- This is a motivation for studying restricted models of computation

Our Picture

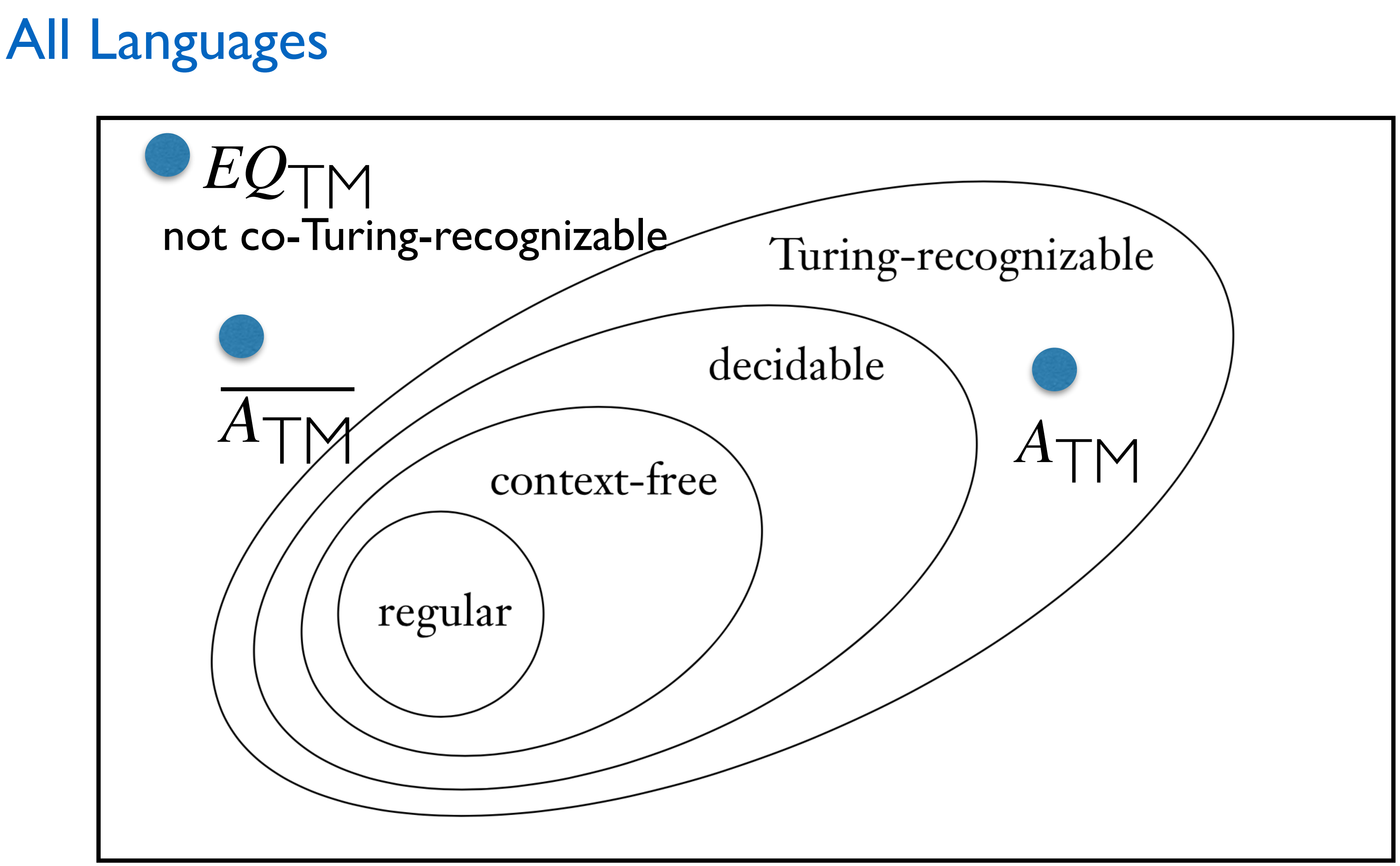
- **Final Question.** Is there a language L such that L is not Turing recognizable and $\overline{L}$ is also not Turing recognizable.
- **Recall.** If $A \leq_m B$ and A is not Turing recognizable, then B is not Turing recognizable.



Class Exercise

- **Theorem.** $\mathrm{EQ}_{\mathrm{TM}}$ is neither Turing recognizable nor co-Turing recognizable (its complement is not Turing recognizable).
- **Proof outline.**
  - To show $\mathrm{EQ}_{\mathrm{TM}}$ is not Turing recognizable, need to reduce a known Turing unrecognizable language to it
  - Show that $A_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$ and $A_{\mathrm{TM}} \leq_m \overline{\mathrm{EQ}}_{\mathrm{TM}}$
  - How does this prove the theorem?
  - Mapping reductions are closed under complement!

Completed Picture of Computability



Complexity Theory

Complexity Theory

- So far, we were focused on computability theory
    - *What problems can and cannot be solved by various models of a computer (starting from most restricted to most powerful)*
- Now, we want to ask the question:
    - *What problem can be efficiently solved by a computer?*
- CSCI 256 covers all about *algorithmic design strategies* as well as analysis tools
    - This class: Assume that you know this and won't focus on it
- Instead focus on classifying complexity of CFGs, TMs, etc as well as reductions to prove problems are NP complete

How to Measure Efficiency

- Time complexity as number of steps
- Complexity measured as a function of input size
- Worst case notion: for any inputs of size n

Definition. Let M be a deterministic Turing machine that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M takes on any input of length n.

Asymptotic Analysis

- As covered in CSCI 256, we don't care about time complexity on small inputs but rather how it grows as n becomes large
- Review asymptotic notation to do this: Big O, Little O

Definition. We say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every $n \geq n_0$:

$$f(n) \leq c \cdot g(n)$$

Definition. We say $f(n) = o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Exercise: True or False?

1. $8n + 5 = O(n)$

2. $1000n + \sqrt{n} = o(n)$

3. $n\sqrt{n} = O(n^2)$

4. $\sqrt{n} = o(n)$

5. $\log_2 n = o(\ln n)$

6. $n \log \log n = o(n \log n)$

Time Complexity Class

Definition. Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The time complexity class, $\text{TIME}(t(n))$, is

$$\text{TIME}(t(n)) = \{L \mid L \text{ is decided by a TM in } O(t(n)) \text{ steps}\}$$

Time Complexity Example

Consider a TM M for for the language $A = \{0^n1^n \mid n \geq 0\}$:

$M =$ "On input string w,

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following if both 0s and 1s remain.
    1. Scan across tape, crossing off a single 0 and a single 1.
3. If either 0 or 1 remains, reject. Otherwise, accept."

- Time complexity?
- Can we do better?

Fun Fact

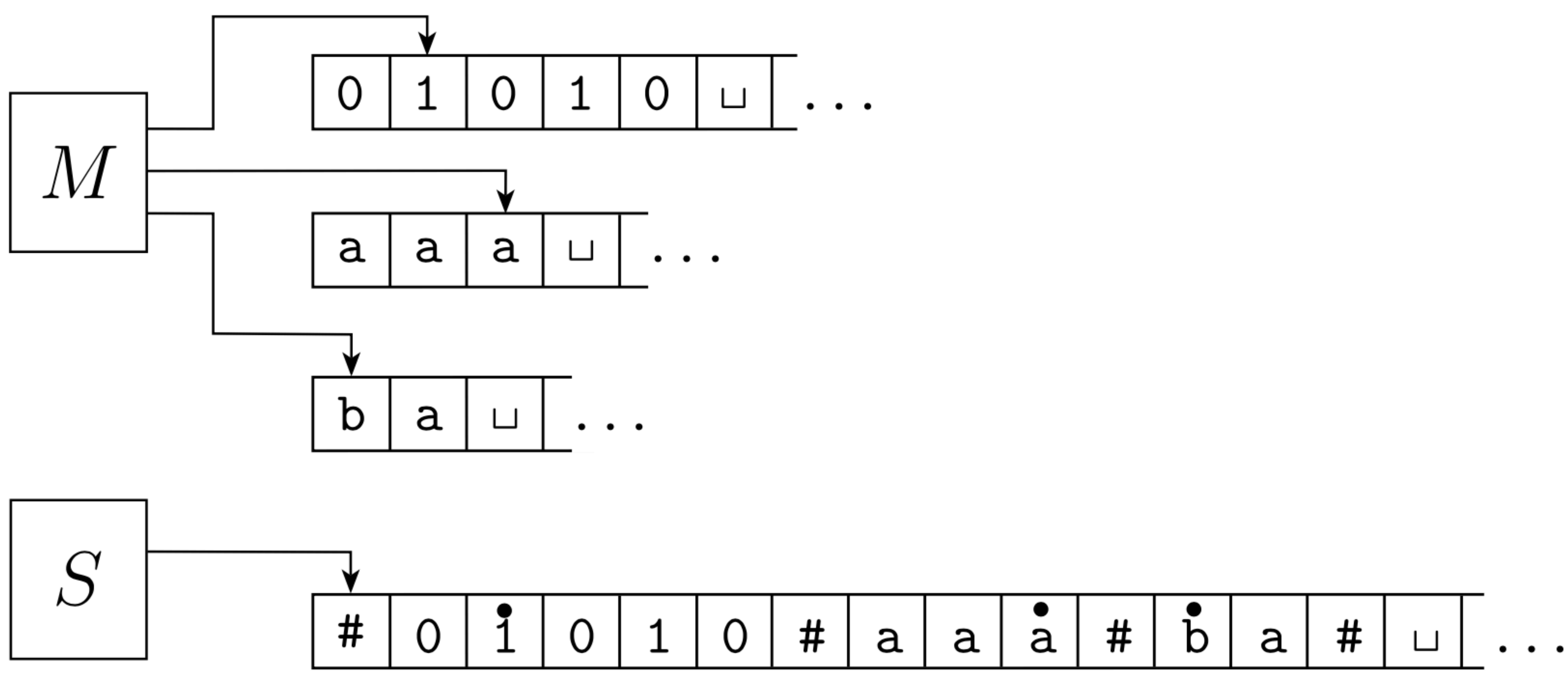
- Let $f(n) = o(n \log n)$. TIME($f(n)$) contains only regular languages!

Two Tapes Can be More Efficient

- How quickly can we decide the language $A = \{0^n 1^n \mid n \geq 0\}$ on a two tape TM?
    - Can do this in $O(n)$ time
- **Takeaway:** Different models of computation can yield different running times for the same language!
- Let's revisit multi-tape TM to single tape reduction with the lens of complexity theory

Multitape TM to Single Tape TM

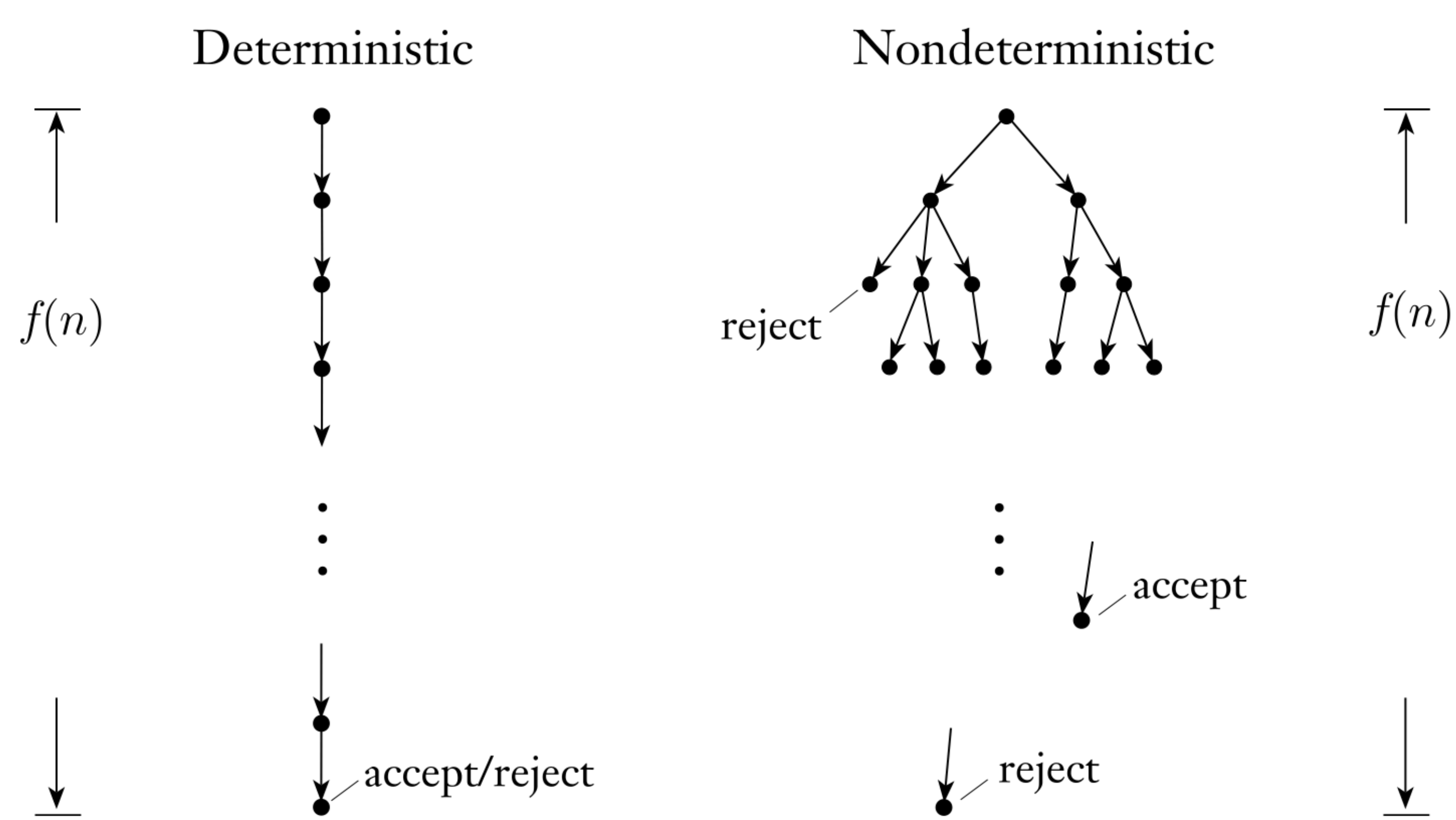
- **Theorem.** Every $t(n)$-time multi-tape TM has an equivalent $O(t^2(n))$-time single-tape TM, where $t(n) \geq n$.



- **Takeaway:** Both models are polynomially-equivalent.

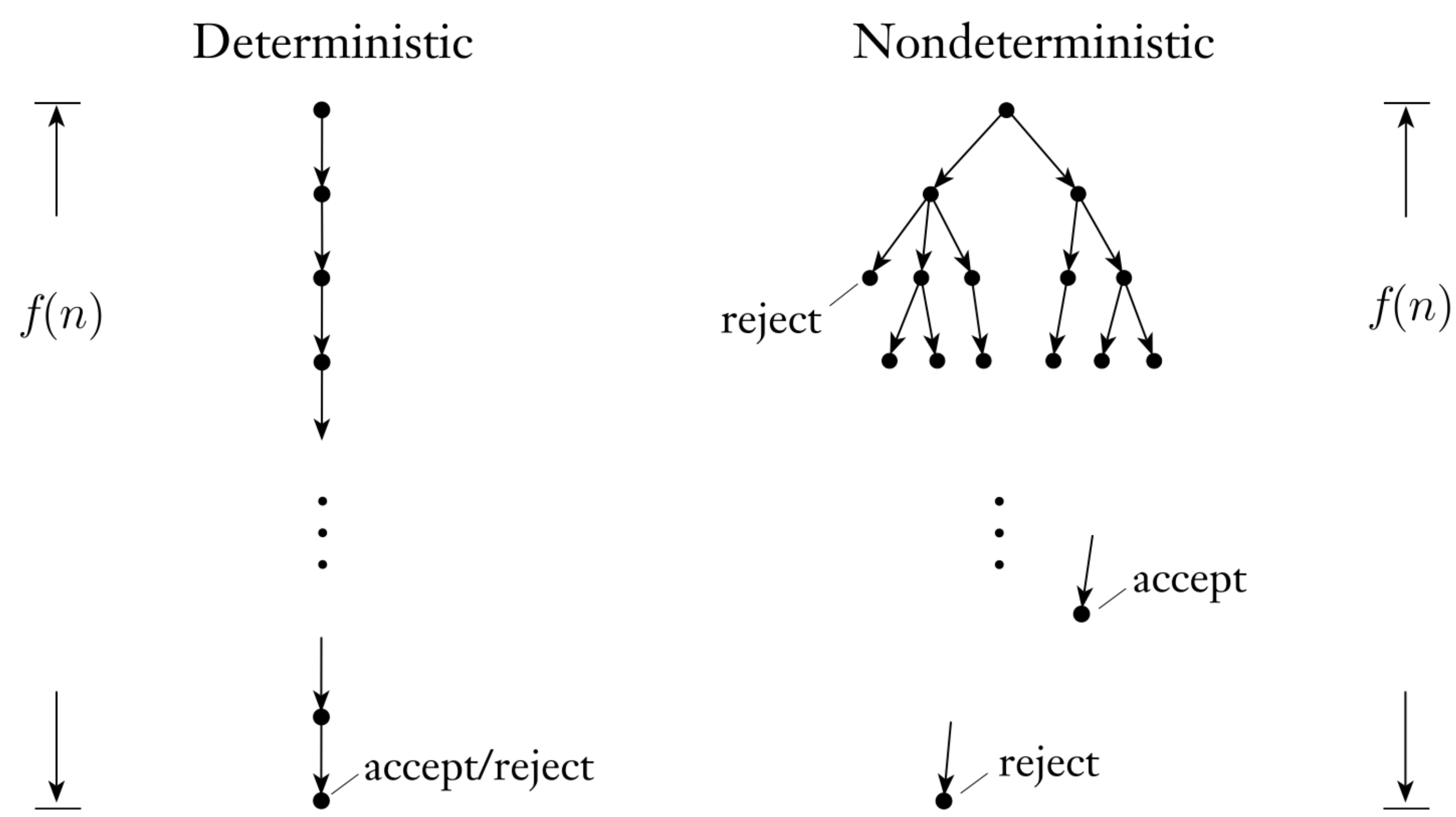
How About Non-Determinism?

- **Definition.** Let M be a non-deterministic TM that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M takes on any branch of its computation on any input of length n.



How About Non-Determinism?

- **Theorem.** Every $t(n)$-time non-deterministic TM has an equivalent $2^{O(t(n))}$-time deterministic TM, where $t(n) \geq n$.



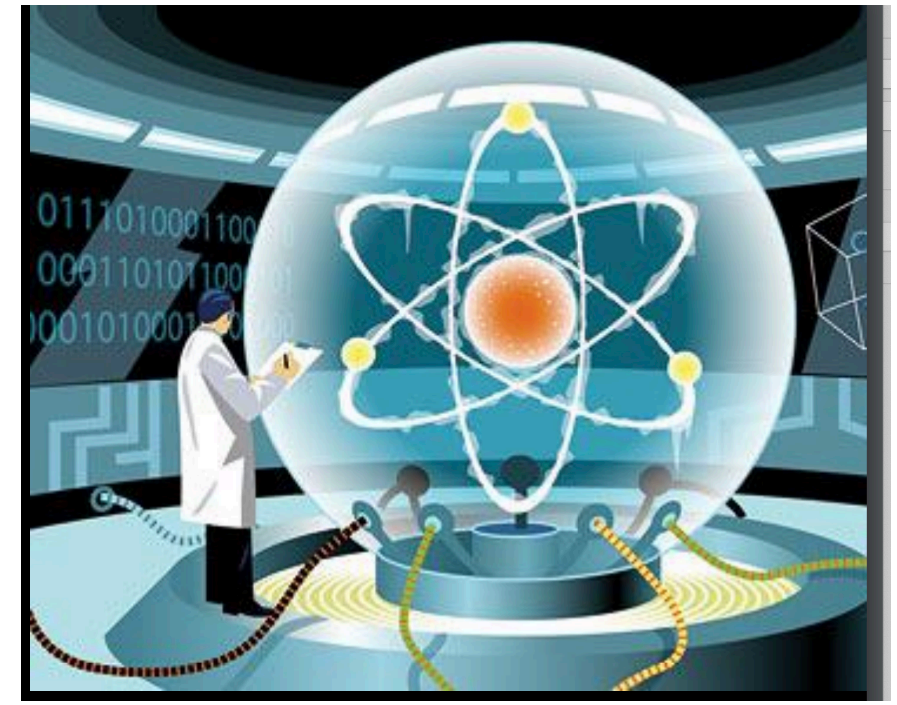
- **Takeaway:** NTM is not polynomially-equivalent to a DTM.

Complexity Class P

Definition. P is the class of languages that are decidable in polynomial time on a single-tape Turing machine. That is,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

Extended Church Turing Thesis



Everyone's intuitive notion of efficient algorithms = polynomial-time algorithms

- Much more controversial:
  - Is $O(n^{10})$ efficient?
  - Randomized algorithms/ quantum algorithms can do much better

Extended Church Turing Thesis

Everyone's intuitive notion of
efficient algorithms
= polynomial-time algorithms

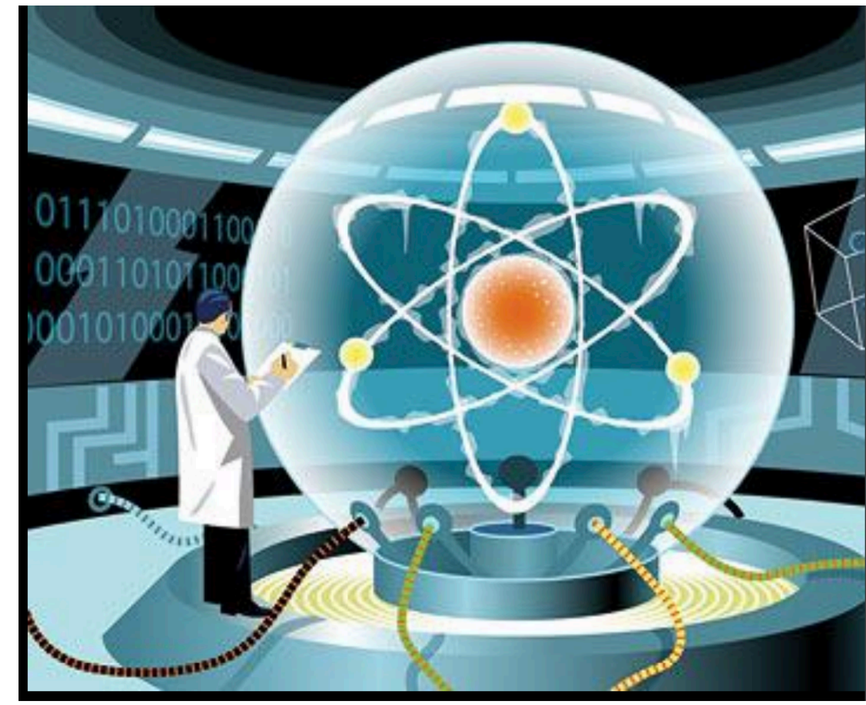


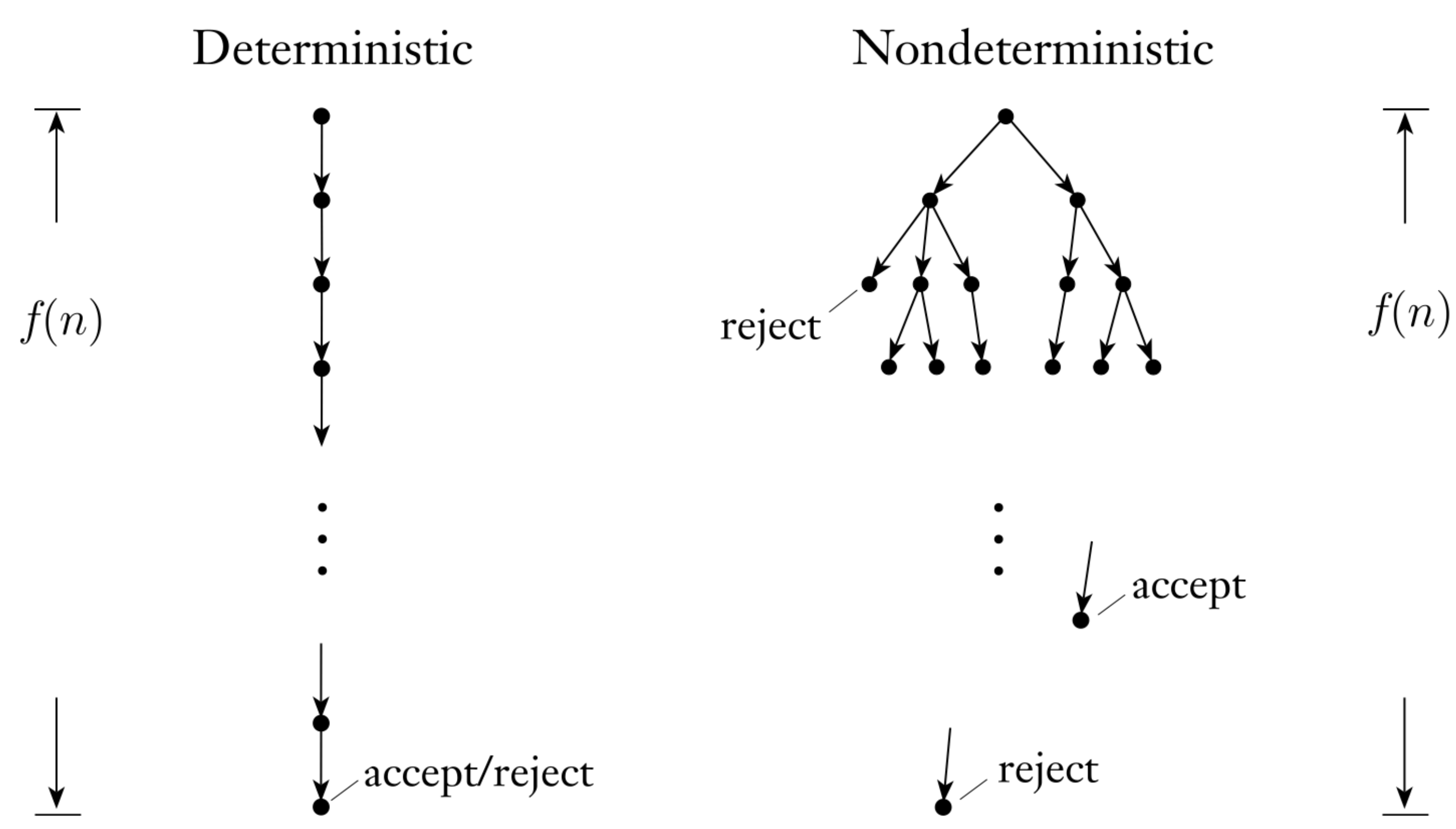
Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Towards NP

- **Definition.** Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The time complexity class, $\text{NTIME}(t(n))$, is

$$\text{NTIME}(t(n)) = \{L \mid L \text{ is decided by an NTM in } O(t(n)) \text{ steps}\}$$



Complexity Class NP

Definition. **NP** is the class of languages that are decidable in polynomial time on non-deterministic Turing machine. That is,

$$\mathrm{NP} = \cup_k \, \mathrm{NTIME}(n^k)$$